

Machine Learning: Community Tools



Dan Guest (HU, Berlin)

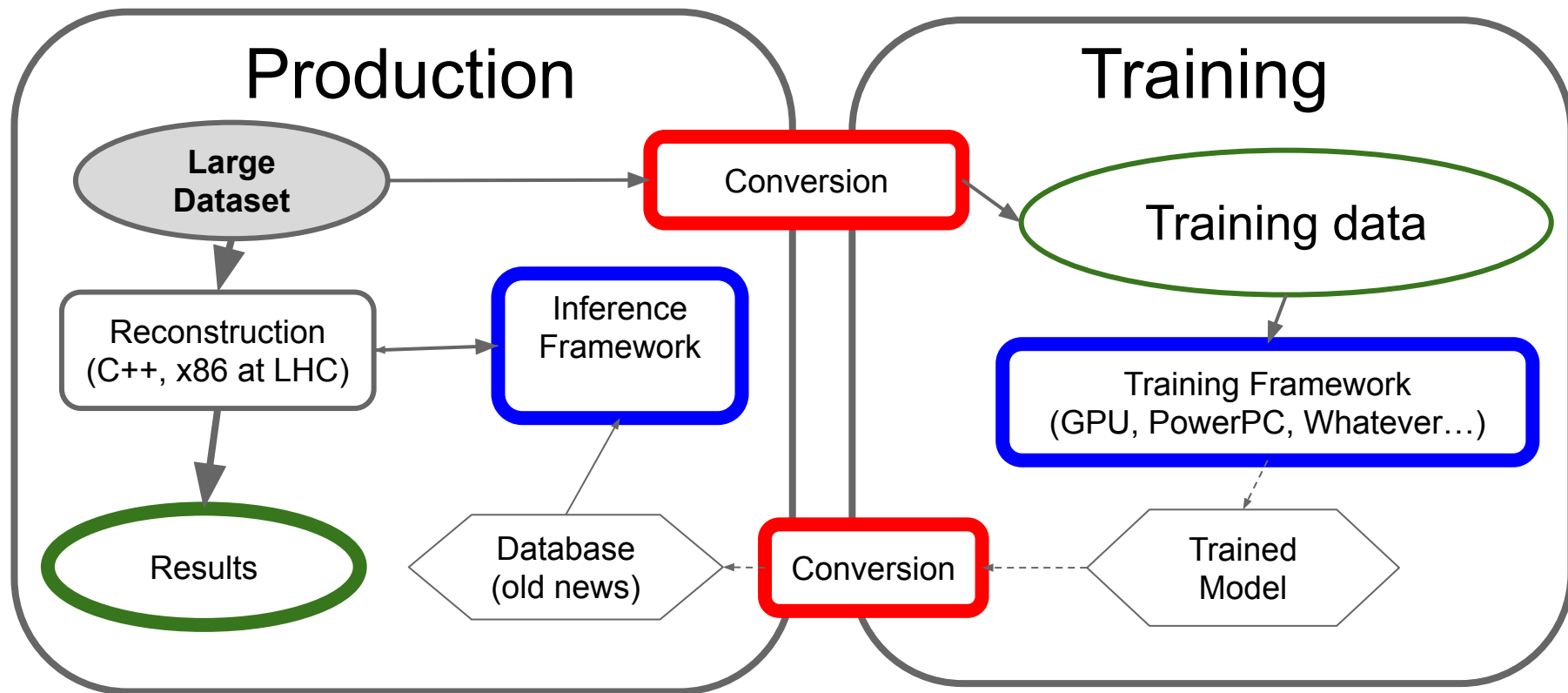
Snowmass Computational Fronteers Workshop

<https://indico.fnal.gov/event/43829/contributions/192876/>

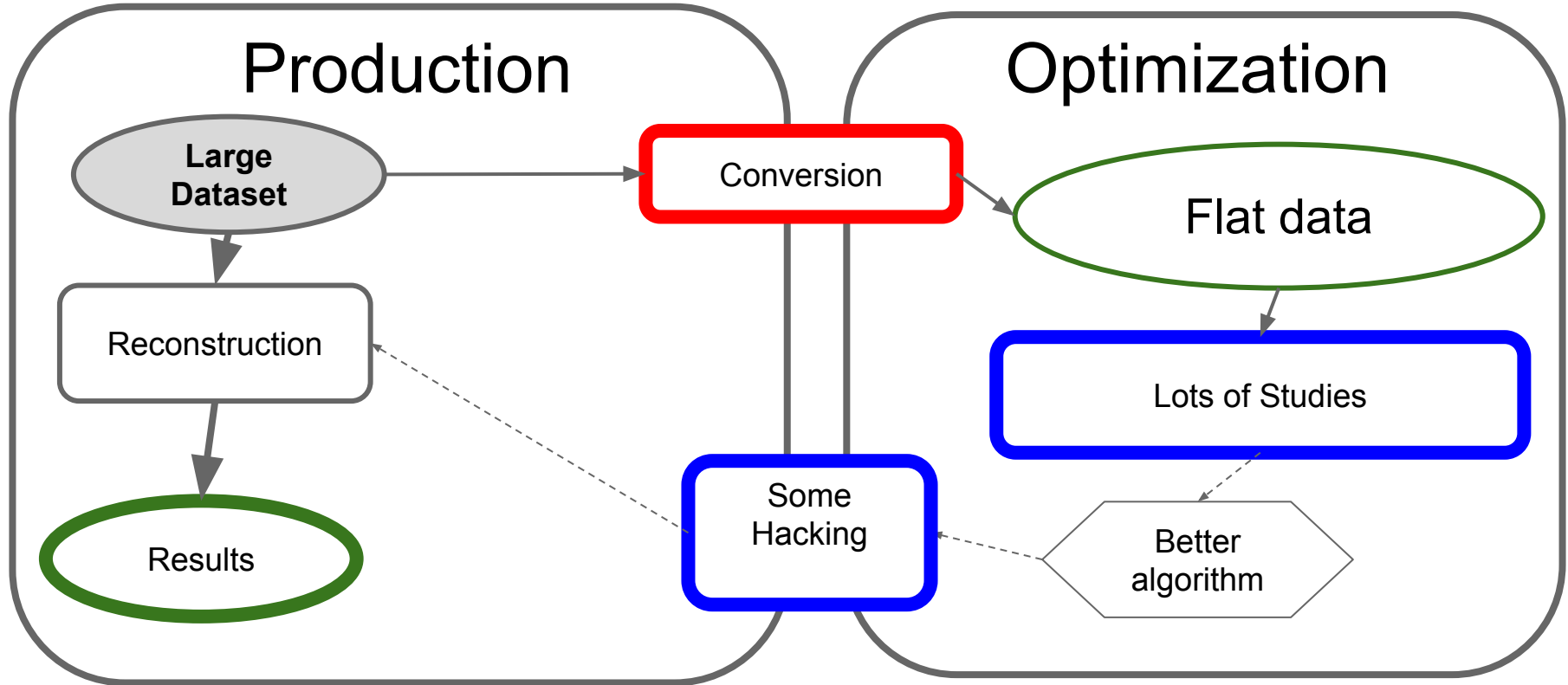
Disclaimer

- I'm on ATLAS, I've been on ATLAS since I wrote my first line of C++
 - Everything here has a huge bias as a result
- I'll talk mostly about neural networks: boosting follows a similar workflow
 - It's still very popular, just not as much development recently
- I'm not an expert in anything I'm about to present
 - I'm a physics postdoc: I mostly think about what *physics* I'll accomplish in the next 5 years

The Machine Learning Workflow

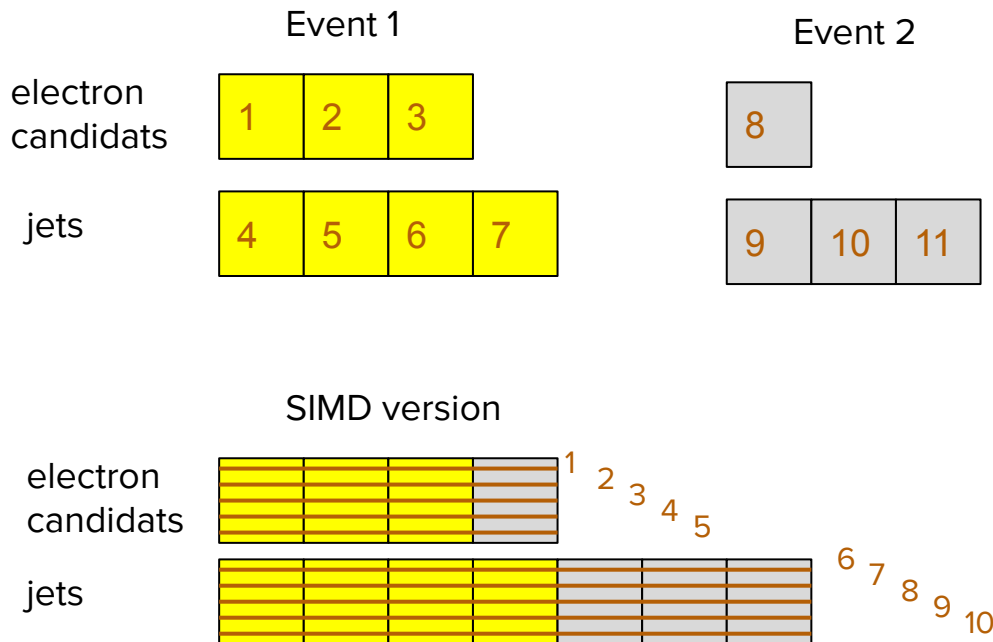


The Non Machine Learning Workflow



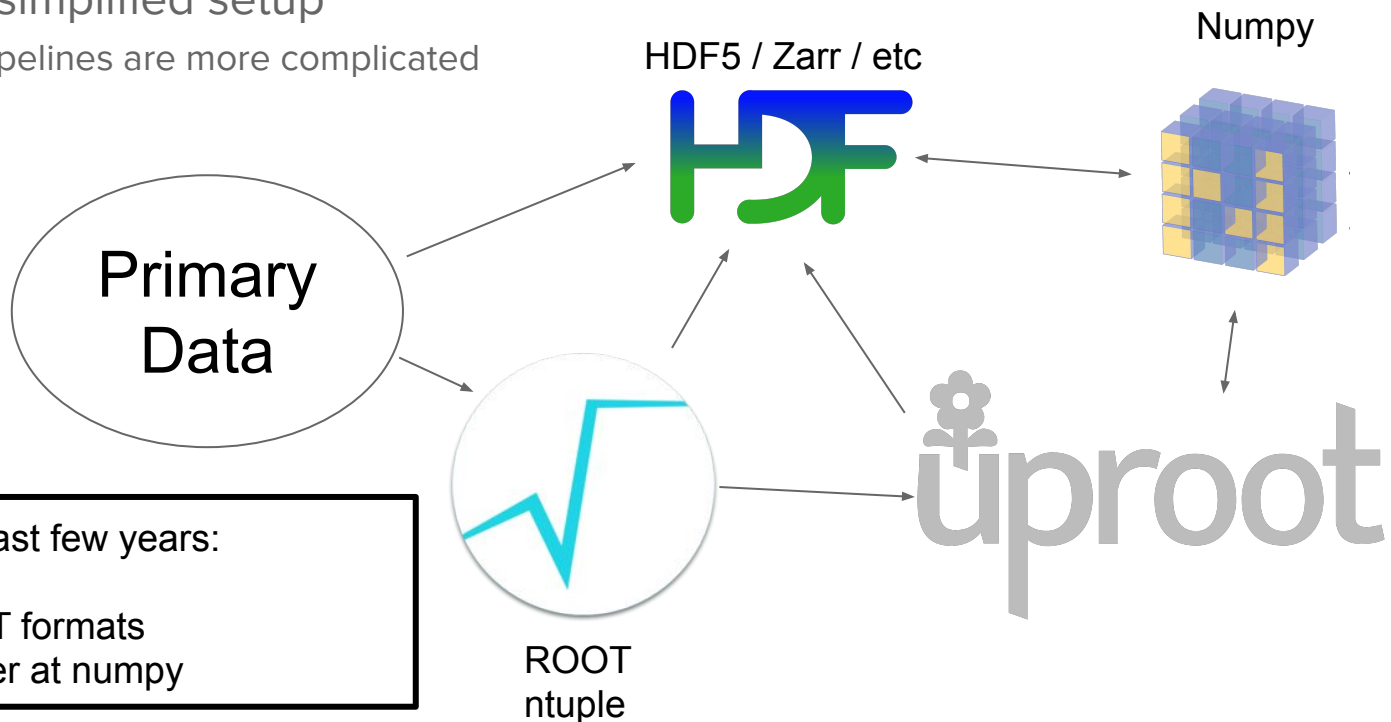
Step 1: restructure primary data

- Traditional hep code: process one object at a time
- ML is driven by parallelism
 - E.g. GPUs
- Our data isn't formatted ideally for SIMD instructions
 - Want arrays not trees
- **Numpy is a more natural fit**
 - For now we're all python



Making Datasets

- **Funtime Activity:** Find Your Way To **Numpy**
- **Note:** oversimplified setup
 - Many pipelines are more complicated

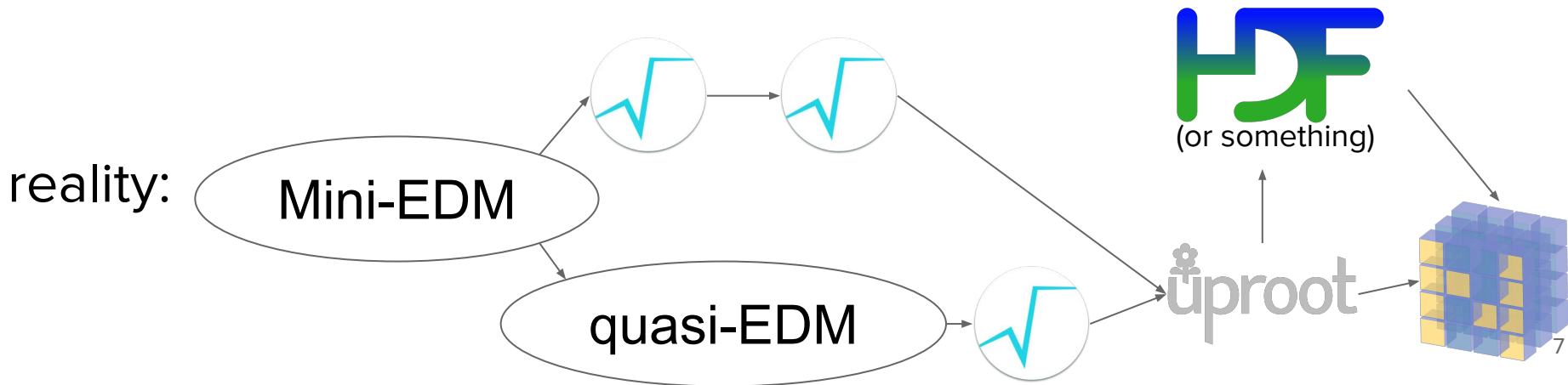
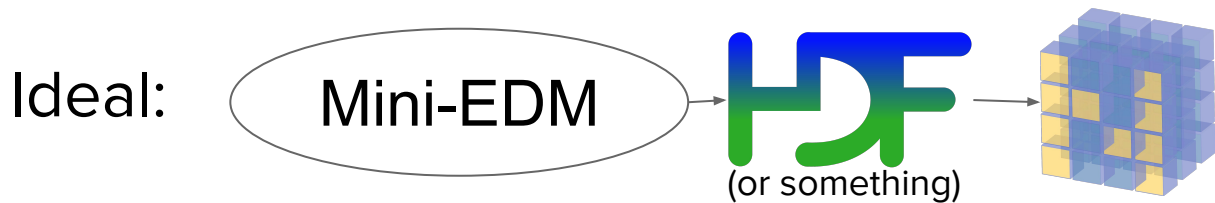


Main changes in the last few years:

- More uproot
- More non-ROOT formats
- People are better at numpy

Can we make this simpler?

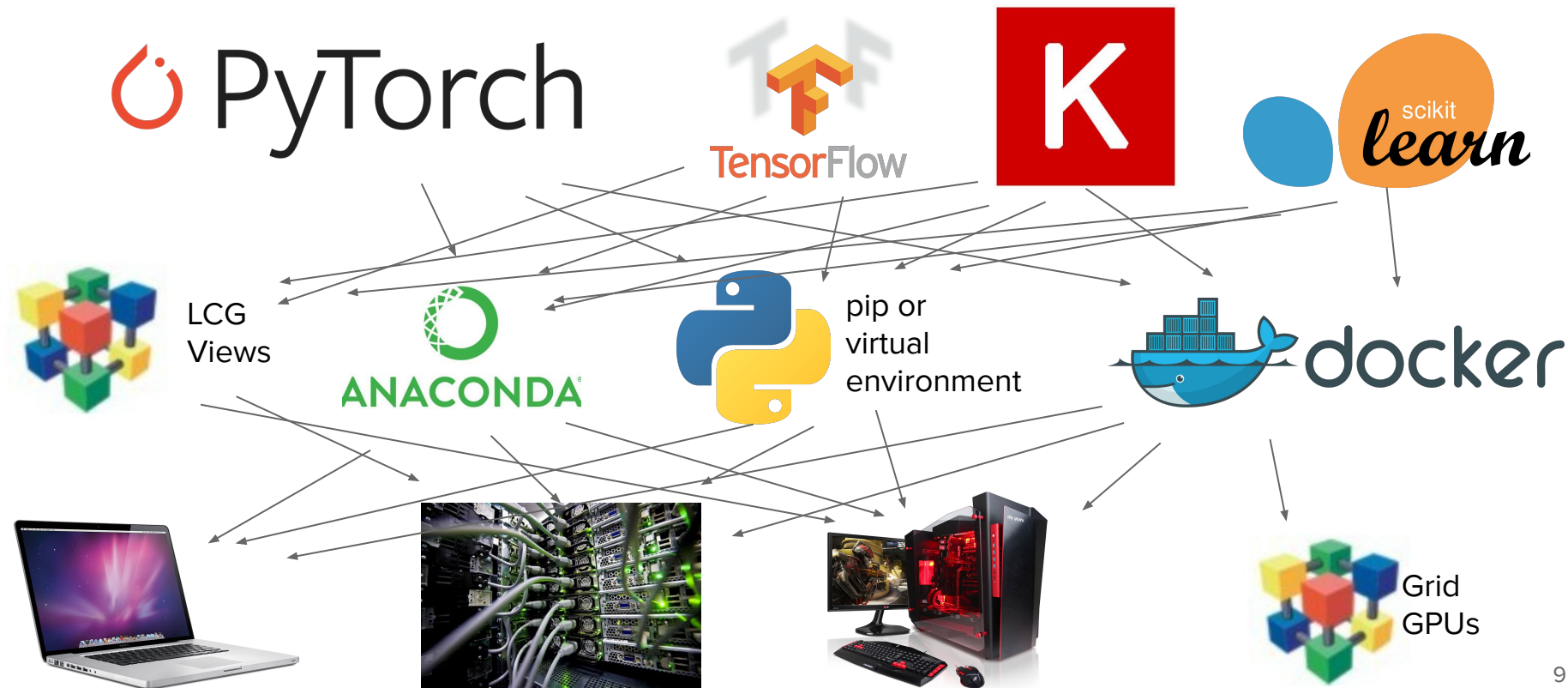
- Maybe... but it's not the obvious place to start
- In practice: Event Data Model (EDM) → training is several steps



But I want to make it better anyway!

- Great! How should we do this?
- LHC: “We have an Exabyte of ROOT data, we should use ROOT!”
 - Problem: our primary data is nothing like our training datasets
 - Also ROOT format := whatever is in a **ROOT file**, not a real standard
- ROOT: “we should invent a new data format!” (RNTuple)
 - Widespread use will require factorization and standardization
- Armchair Data Nerds (i.e. me): “Hasn’t someone already solved this problem?”
 - Parquet is another promising candidate
 - Many open source projects are receiving little to no funding!
- **My opinion: we need more core developers for experiments**
- **Anyway, once we have some training data...**

A few choices: Training “Flowchart”



How to improve? Focus on distribution

	pip / venv	conda	LCG Views	Containers (Docker, podman, singularity)
User configurable	Y	Y	X	Y
Work on laptop	Y	Y	:('	Y
Work on desktop	Y	Y	Y	Y
Works on cluster	:('	:('	Y	Y (need work)
Works on grid	:('	:('	Y	Y (need work)
Composable	Y	Y	:('	X

- **LCG views:** lazy load an OS from a FUSE mount
 - Requires network!
- **Containers:** eager-load an OS from an image
- Containers could be synched / distributed more intelligently
 - [Examples](#) exist
 - Consider podman?
 - Or native docker?
- I assume root access to a desktop, normal user privileges for a cluster

- Containers can be user-defined, but we could centralize support (e.g. base layers)


How do we apply models?

Training

Database

Inference

Symmetric


TensorFlow

 **protobuf**
Protocol Buffers


TensorFlow

Factorized


PyTorch

{JSON}



ONNX

lwttnn

 ONNX
RUNTIME

But isn't symmetric easier?

Advantages of symmetric:

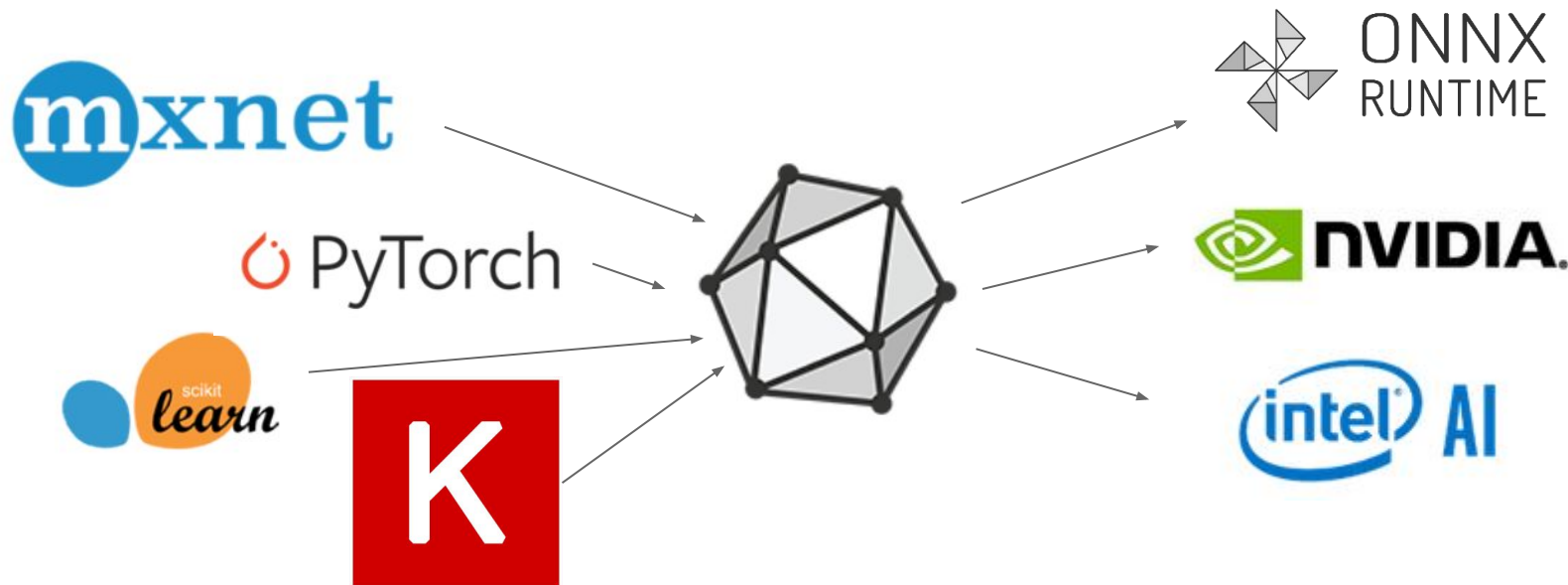
- **Only one library**
- Cutting edge support
 - No need for standards
- Less validation required
- No need to understand format(s)

Problems with symmetric:

- Framework bloat
 - **CMSSW has ~4 NN libraries**
 - Dependency hell [\[1\]](#)
- C++ binding support

- Eventually, symmetric implementations **tend to become factorized**
 - E.g. Tensorflow in CMSSW
- **But factorization will require model translation**

Will ONNX save the day?



- Pro: there's a [well defined specification](#), with many community contributors
 - Brought to you (in part) by Microsoft. What a time to be alive!
- Con: common standards always lag, **big players have their own inference frameworks**

Or will industry push lock-in?

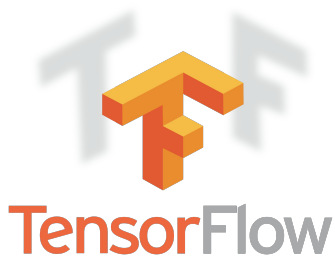
 PyTorch

VS

**?
=**



VS



- Current trend is toward a duopoly: Will ML go the way of instant messaging?
 - Can we help?

In short:

It's (mostly) chaos!

(i.e. like the rest of academic code)

Maybe chaos (autonomy) isn't so bad

- We work in small teams
 - One reality of our funding structure
- Every application is different
- Commonality within experiments:
 - Branding
 - Raw ingredients
 - Internal quality review
- Think of it as a franchise

Wikipedia: "...a franchisor licenses its know-how, procedures, intellectual property, use of its [business model](#), brand... In return the franchisee pays certain fees and agrees to comply with certain obligations, typically set out in a Franchise Agreement."



Summary: What can we do to help?

- Keep a balanced menu
 - Best practices can evolve, especially in ML
 - Avoid monoculture
- Know what you eat
 - Experiments can't avoid having ML engineers
 - Especially with more integrated ML
 - Common standards help here
- Pay your workers (anything)
 - Open source software doesn't pay (much)
 - But we rely on it heavily
 - **Even citing your libraries helps (and it's free)**



Thank You!



**No, Really... it's
over now. (backup after this)**

So what's new in the ML workflow?

- **At a high level:** very little
- Making derived datasets → 50% of every PhD
 - **Training datasets aren't very big:** size < 1 TB
- Running stand-alone optimizations → 25% of many PhDs
- Calibration data:
 - We've done this forever
 - In many cases we're overtooled here
- **At a low level:** lots of things are **new to physicists**
 - **But don't confuse "new to physics" with "new technology"**
- **Bottom line:** this *should* be easy **as long as we keep it simple**

The boring spectrum

Standardized	Stable	Developing	Bleeding edge
ONNX	ROOT files	TMVA	RNtuple
Parquet	EDMs	Tensorflow, Keras	Data Lakes
HDF5	CVMFS	PyTorch, libtorch	
JSON		uproot	

ATLAS inference notes

- [lwttn](#): stable, probably won't change much in the near future
 - This is a good thing: it will go into run 3 trigger (tau, b-jet) and (hopefully) tracking
- Onnx runtime: [merged to AtlasExternals](#)
 - It's 11 MB or so: any reason **not** to merge this?
 - Supports many more models than lwttn
- [MVAUtils](#): **BDT support** in reconstruction
 - Should probably be a stand alone library
 - TMVA dependency recently removed, could we remove ROOT?
 - Support for **XGBoost, LightGBM, TMVA**
- Overall: **we don't need full training libraries in reconstruction**
 - Software team is small, support isn't worth our time

Training Side

- Strategy is the same as ever
 - No “hep” code
 - Use docker containers
- Build system:

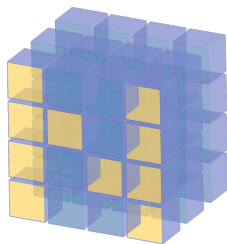
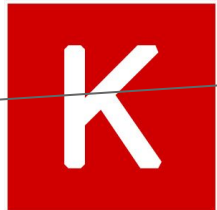
```
pip install -r requirements.txt
```

- Lots of work scaling up
 - Cern has a GPU batch now!
 - Adding more grid sites
- Everything is python
 - **The grad students love python**

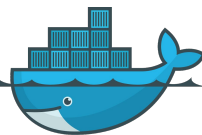


Also: we still need containers

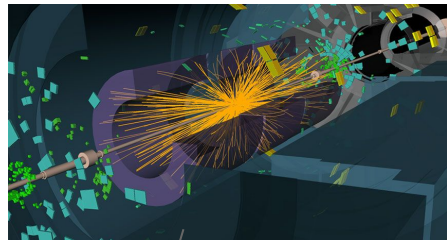
- Small scale testing: use your laptop or local cluster
- Medium scale: **lots of options, see [Doug's talk](#) (also CERN batch)**
- Large scale: submit to the grid
 - See [instructions](#) and [tutorial](#)



NumPy

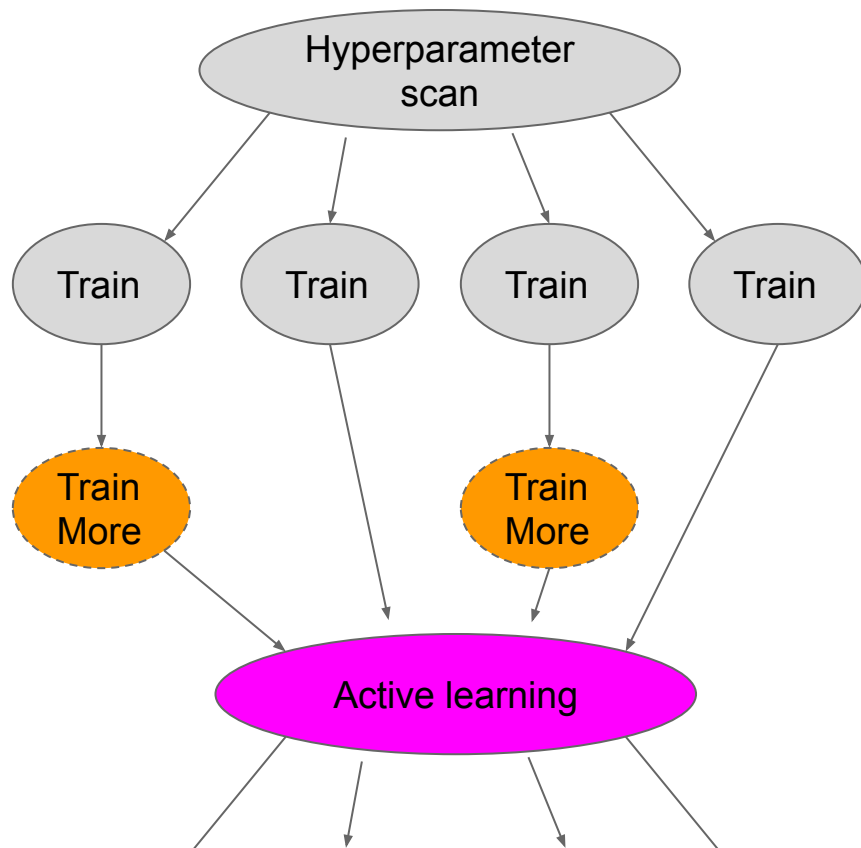


docker



Are we limited? Not so much (anymore)

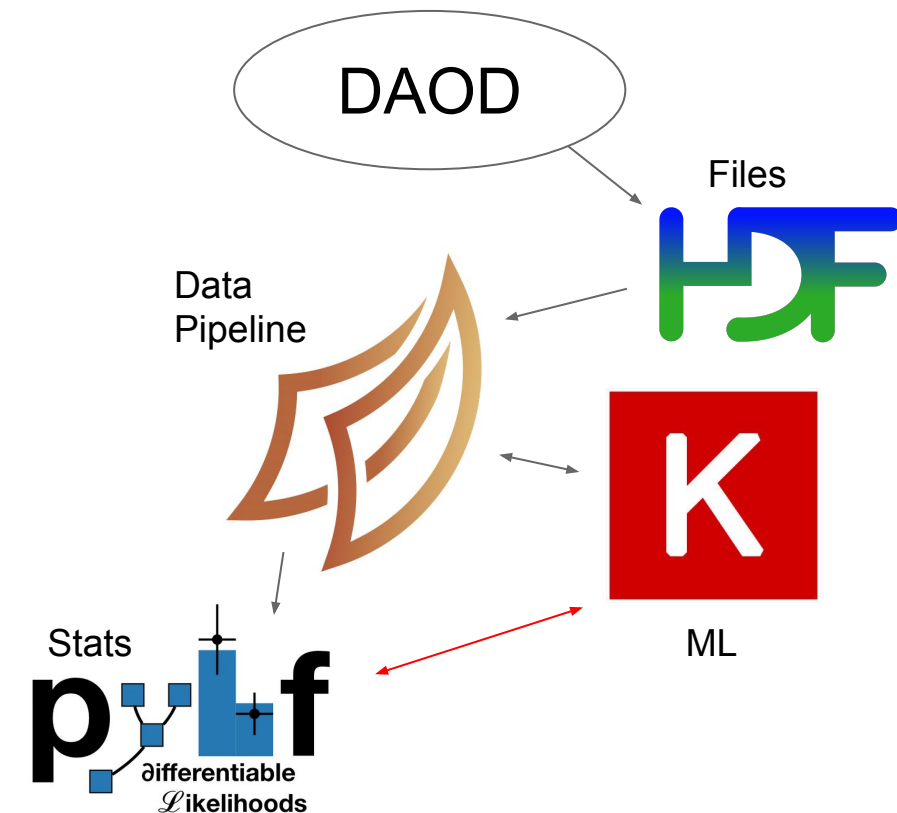
- Our NNs are pretty simple
 - We built a pretty good detector
 - Complexity increase not certain
 - [FTAG RNN](#) (21 MB file): 666,462 pars
 - Many industry nets are $O(10M)$
- Our jobs aren't too complicated
 - HP scans work “natively” on the grid
- We could use an “extension” mechanism for long jobs
- Other ideas (active learning) *could* come later



Crazier Ideas

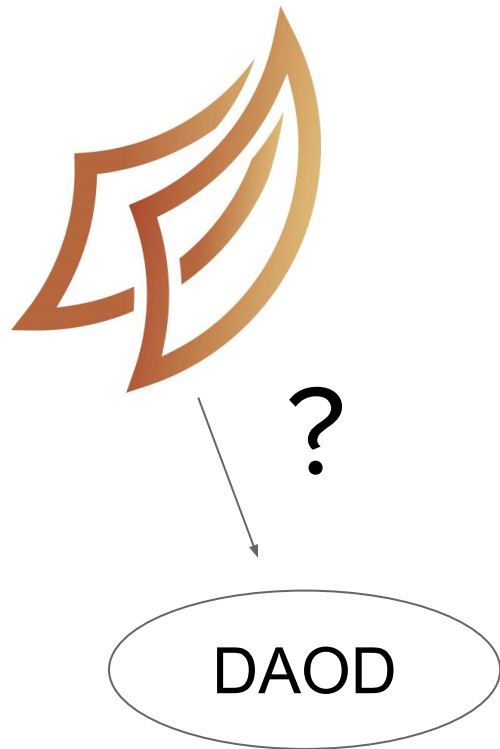
Python: fun for the whole analysis!

- Write a for loop over PHYSLITE
 - Apply systematic, dump dataset
- Move everything into python
 - ML: anything you want
 - Lots of data formats
 - Use pyhf / scipy for limits
 - **end-to-end learning is “easy”**
 - **d(limit)/d(anything)**
- **Vectorized out of the box**
- This assumes
 - I know what I’m doing physics-wise
 - **I don’t want to write *any* tools**



Can we use this for production?

- **No (not yet)**
 - no simple way to *write* a (D)AOD
 - Have to deal with awkward arrays
- **Maybe it's worth trying**
 - We could solve the inference problem
 - Or at least push it upstream
 - We could **vectorize on HPCs** for “free”
- Need python bindings for our EDM
 - PyROOT doesn't count
 - Bindings need to be batch-wise



What would be nice

- Ambitious goal: read AOD easily

```
pip install atlas_edm
```

- Or as an intermediate step

```
yum install atlas_edm
```

- We already have AnalysisBase as a docker image
 - But it's O(1 GB)
 - Also installs a lot (compiler, ROOT)

Roadmap: Less (C++) is more

- Figure out what the EDM needs to depend on
 - Maybe rip a few things out of ROOT, i.e. TLorentzVector
 - CLHEP / Eigen for particles / tracks
 - FastJet probably needed for jets
- Remove ROOT
 - Replace it with uproot (the C++ bindings)
- *Then* work on python bindings
- Having a “light” release would be nice for a few reasons
 - Weird HPC architecture
 - ML (obviously)
- **Open data / education deserves its own mention**

What about systematics?

- The current paradigm: **take your data, leave the HEP ecosystem**
 - Not a great way to use common code (or produce it)
- A lot of CP code is of the form
 1. Get a jet
 2. Look up variation by bin
 3. Multiply jet by variation
- Basically `pt *= scale_factor[np.digitize(pt, pt_bins)]`
 - **Again, vectorized out of the box, can hook in GPU**
- Missing parts
 - Community: hardly anyone actually does this
 - Tools are *slightly* more complicated than this
- **We could do most post-PHYSLITE analysis in python**

How big are the ATLAS libraries?

Just the libxAOD .so files (excluding Dict files):

```
du -s /usr/AnalysisBase/21.2.108/InstallArea/x86_64-centos7-gcc8-opt/lib/libxAOD*.so | grep -v Dict.so | awk '{a+=$1} ; END {print a}'
```

18,288

Every .so file

```
du -s /usr/AnalysisBase/21.2.108/InstallArea/x86_64-centos7-gcc8-opt/lib/*.so | awk '{a+=$1} ; END {print a}'
```

113,164

How big is ROOT?

```
du -s  
/usr/AnalysisBaseExternals/21.2.108/InstallArea/x86_64-centos7-gcc8-opt/lib/lib* |  
egrep 'lib[A-Z]+[a-z]*' | awk '{a+=$1}; END {print a}'
```

247,304